Lancer de rayon sur des architectures parallèles : une étude de performance

Thierry Priol, Kadi Bouatouch

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N°973

Programme 2

LANCER DE RAYON SUR DES ARCHITECTURES PARALLELES : UNE ETUDE DE PERFORMANCE

Thierry PRIOL
Kadi BOUATOUCH

Février 1989

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Lancer de rayon sur des architectures parallèles : une étude de performance

Ray-tracing on parallel computers: a performance study

Thierry Priol
Kadi Bouatouch
IRISA *
Campus de Beaulieu
35042 Rennes Cedex
France

Publication Interne n°451 - Janvier 1989 - 20 Pages

Résumé

Nous présentons un algorithme parallèle de lancer de rayon adapté à des architectures distribuées dont les processeurs échangent des données par l'intermédiaire de messages. L'algorithme de lancer de rayon est particulièrement coûteux en temps de calcul mais également en mémoire. La parallélisation d'un tel algorithme doit permettre la meilleure utilisation à la fois des ressources de calcul et de la mémoire. Nous décrivons un découpage spatial du volume englobant de la scène à synthétiser, permettant ainsi d'associer à chaque processeur, un sous-espace de la scène. La taille de chacun de ces sous-espaces est déterminée par une étape de prétraitement consistant en un sous-échantillonnage de l'image. Les pixels de ce sous-échantillonnage sont traités afin d'estimer la charge de travail de chaque processeur. Nous présentons les résultats de l'exécution de cet algorithme sur un hypercube iPSC/2.

Abstract

A parallel ray-tracing algorithm for parallel distributed memory computers is presented. Ray-tracing involves a lot of computations and requires large memory. We describe, in this article, a subdivision scheme which creates as many 3D regions as processors. Static load balancing is performed by sub-sampling the image. Some results of such parallel ray-tracing algorithm on a multiprocessors hypercube iPSC/2 are presented.

1 La méthode du lancer de rayon

Parmi les algorithmes de rendu utilisés dans le domaine de la synthèse d'image, la méthode du lancer de rayon est la seule qui permet de créer des images d'un grand réalisme (ombrage, réflexion, transparence). Elle simule le fonctionnement d'un appareil photographique, en suivant le trajet inverse de la lumière. Un plan virtuel composé de pixels et représentant l'écran est placé entre l'observateur et la scène à synthétiser. L'algorithme

*Institut de Recherche en Informatique et Systèmes Aléatoires.

consiste à déterminer l'intensité de chaque pixel en fonction de lois photométriques établies. Pour chaque pixel de l'écran, un rayon passant par celui-ci et issu de l'observateur permet de déterminer les surfaces visibles. Si le rayon intersecte un objet, plusieurs rayons sont alors nécessaires pour connaître la valeur de l'intensité associée au pixel. Il faut en effet déterminer si le point d'intersection est ombragé ou non, ce qui nécessite la génération d'un rayon vers chaque source lumineuse. D'autre part suivant les qualités photométriques de l'objet intersecté, des rayons dans les directions réfléchie et transmise sont également nécessaires pour rendre l'aspect réfléchissant ou transparent de l'objet intersecté. Au total, la synthèse d'une image nécessite plusieurs millions de rayons pour lesquels ce traitement doit être effectué.

La scène à synthétiser est modélisée par un arbre CSG [1]. Le CSG permet d'effectuer des opération booléennes sur les volumes de primitives simples. Ainsi, un objet est modélisé par un arbre binaire dans lequel les nœuds représentent les opérations ensemblistes (union, intersection, différence) effectuées sur les volumes, et dans lequel les feuilles sont les primitives volumiques élémentaires (sphère, parallélépipède, cylindre, etc...).

Pour calculer le point d'intersection réel entre un rayon et un objet de la scène, il est nécessaire de parcourir l'arborescence de haut en bas pour déterminer tous les points d'intersection avec les primitives, puis de bas en haut pour connaître les points d'intersection effectifs calculés en fonction des opérateurs CSG. Sur des scènes importantes (plusieurs milliers d'objets) cette opération est très coûteuse.

Sans optimisation, plusieurs jours sont nécessaires pour calculer une image de complexité moyenne (1000 primitives) sur une station de travail de type SUN/3.

2 Vers une diminution des temps de calcul

De nombreux travaux ont permis de diminuer de façon significative le temps nécessaire au calcul d'une image. La plupart de ces travaux concernent l'optimisation des algorithmes afin de réduire le volume de calcul. D'autres travaux décrivent des méthodes pour réduire le temps de calcul en utilisant au mieux des architectures parallèles. Dans ce domaine, les réalisations sur des machines composées de plusieurs dizaines de processeurs ont portées le plus souvent sur des algorithmes peu efficaces mais facilement programmables. En effet, la programmation distribuée est difficile à cause du langage qui n'est pas souvent adapté à l'algorithmique du lancer de rayon, du manque d'outils de mise au point et enfin de la taille mémoire disponible dans chaque processeur (en général 128-512Ko). De nouvelles machines, récemment disponibles, diminuent sensiblement ces contraintes (Ametek 2010, Intel iPSC/2). La parallélisation d'algorithmes plus complexes peut donc être envisagée et expérimentée. Afin de situer nos travaux dans le contexte général, nous faisons le point sur l'optimisation séquentielle des algorithmes puis sur l'utilisation des nouvelles architectures parallèles pour le lancer de rayon.

2.1 Optimisation des algorithmes séquentiels

Le traitement le plus coûteux dans un algorithme de lancer de rayon est la recherche des intersections entre rayons et objets puisqu'elle représente jusqu'à 80% du temps de calcul d'une image. Les optimisations ont donc naturellement porté sur la diminution du nombre de calculs d'intersection. De nombreuses solutions algorithmiques ont été proposées. Parmi celles-ci nous décrivons les deux techniques les plus efficaces : la hiérarchisation des volumes englobants et la subdivision spatiale du volume initial de la scène.

[1] Constructive Solid Geometry

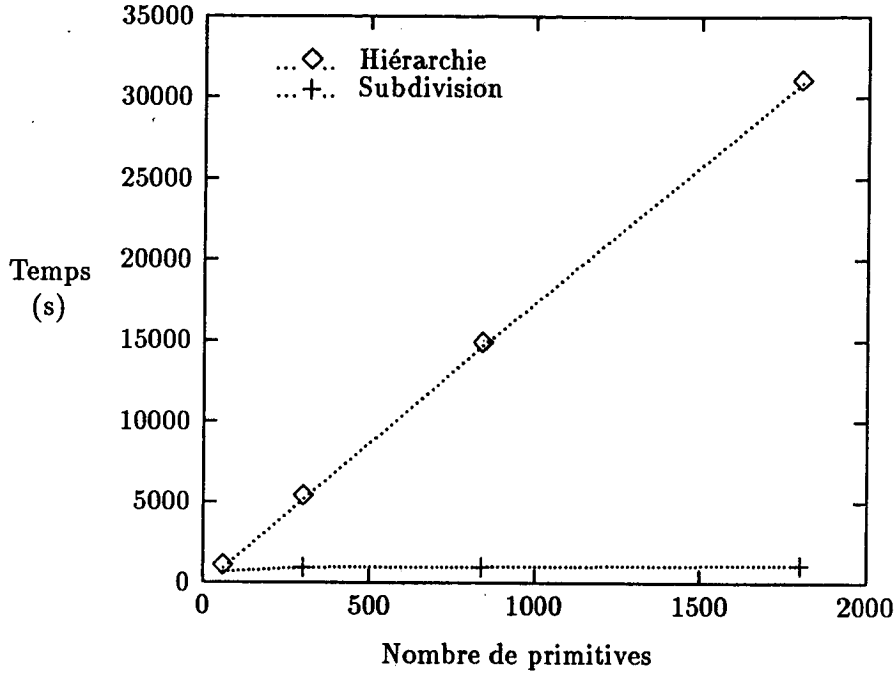

Figure 1 : Temps de synthèse.

La *hiérarchisation des volumes englobants* permet de minimiser le nombre de calculs d'intersections entre un rayon et les objets composant la scène. Elle consiste à associer à chaque nœud de l'arbre CSG un volume qui contient tous les objets appartenant au sous-arbre de ce nœud. Ainsi, lors de la recherche des objets intersectés par un rayon, si celui-ci n'intersecte pas le volume englobant associé à un nœud, il est inutile de parcourir le sous-arbre de ce nœud [21]. Par contre, le parcours de l'arborescence de bas en haut reste nécessaire afin de déterminer le point d'intersection en fonction des opérateurs CSG.

La technique de *subdivision de l'espace* permet également de réduire le nombre de calculs d'intersection en évitant le parcours de l'arbre CSG global. Le volume initial de la scène est découpé en régions. Lors de la synthèse d'une image, les rayons traversent les régions et seuls les objets contenus dans les régions intersectées sont examinés. Plusieurs méthodes de subdivision ont été proposées et diffèrent par l'algorithme de découpage ainsi que par l'algorithme permettant de passer d'une région à l'autre lors du lancer de rayon [1, 14, 15, 18, 23]. Une technique de restriction d'arbre CSG [1] permet d'associer à chaque région issue du découpage un sous-arbre CSG plus petit que l'arbre CSG initial, minimisant ainsi les parcours d'arbre.

2.1.1 Conclusion

Les performances de ces deux techniques sont résumées par les figures 1 et 2. La technique de subdivision spatiale permet d'obtenir des temps de traitement à peu près constants pour des images possédant 100 à 2000 objets. Pour les algorithmes utilisant une hiérarchisation des volumes englobants, les temps de calcul sont beaucoup plus importants et au mieux linéaires en fonction du nombre d'objets. En effet, ces résultats ont été obtenus à partir d'une image ne possédant que des opérateurs d'assemblage qui sont les plus simples à évaluer lors du parcours de l'arborescence. Des facteurs d'accélération de l'ordre de 30 ont été obtenues avec des techniques de subdivision spatiale dès lors que l'image possède

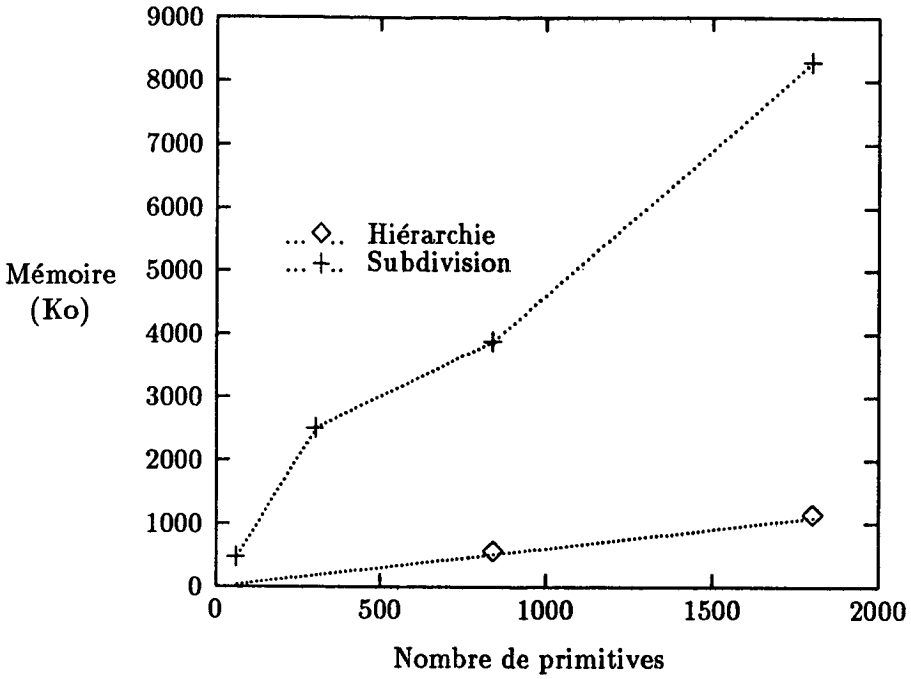

Figure 2 : Taille mémoire.

plus de 1000 primitives. En contrepartie, la subdivision spatiale nécessite beaucoup plus de mémoire pour sauvegarder les régions créées par le découpage ainsi que les sous-arbres CSG associés car une certaine redondance d'information existe. En effet un objet peut appartenir à plusieurs régions simultanément. En conclusion, la technique de subdivision spatiale apparait comme étant la seule permettant de synthétiser des images complexes (plus de 1000 primitives) en un temps raisonnable.

2.2 Parallélisation des algorithmes

Quelques méthodes ont été proposées pour utiliser des architectures multiprocesseurs à mémoire distribuée, suivies pour certaines d'entre-elles d'une expérimentation. Deux types de parallélisation ont été proposées:

1. Duplication des données dans chaque processeur.
   Cette technique décrite dans [2, 20] a été mise en œuvre sur les machines CRISTAL et LINKS.

2. Partition du volume initial de la scène.
   A chaque processeur est affecté un sous-volume issu du découpage du volume initial de la scène [5, 11, 19]. Cette méthode soulève un grand nombre de problèmes liés à la gestion du parallélisme ce qui freine sa mise en œuvre réelle. C'est pourtant de cette technique qu'on peut espérer les meilleurs temps de calcul pour des scènes complexes.

Dans les paragraphes suivants, nous décrivons brièvement ces deux types de parallélisation, puis nous introduisons notre algorithme de lancer de rayon mis en œuvre sur un hypercube iPSC/2 qui utilise une méthode de partitionnement.

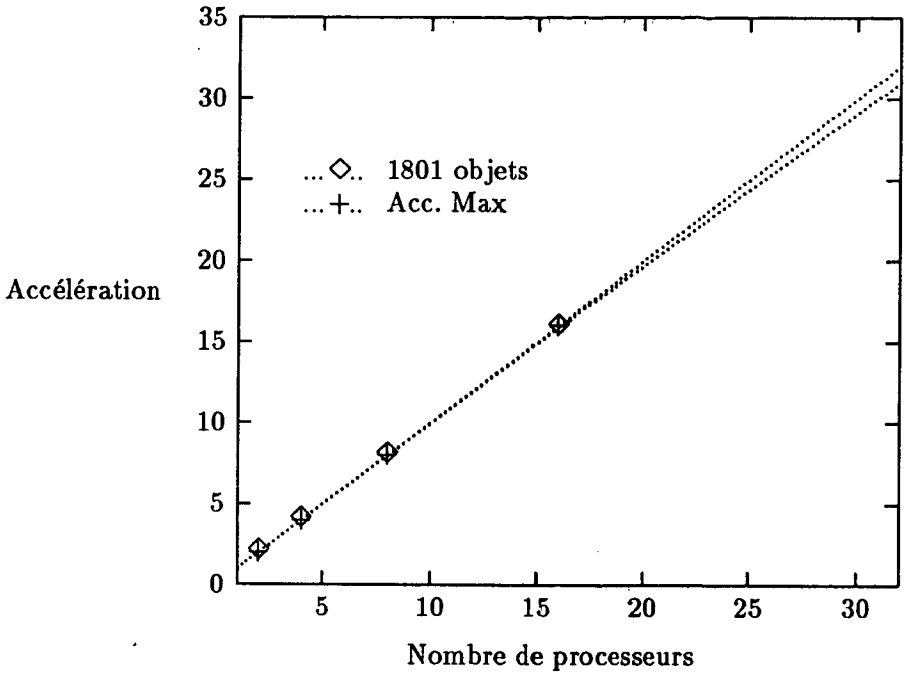

Figure 3 : Accélération de l'algorithme de lancer de rayon avec duplication des données

2.3 Duplication des données

La parallélisation la plus courante de l'algorithme de lancer de rayon résulte directement du principe même de l'appareil photographique où la pellicule est impressionnée simultanément par plusieurs rayons lumineux. Chaque pixel de l'écran peut donc être traité comme une entité indépendante. La parallélisation consiste alors à dupliquer la base de donnée initiale dans chacun des processeurs et à associer à chacun d'eux un groupe de pixels à calculer correspondant à une partie de l'écran. Le groupe de pixels peut être soit un pixel soit une ligne de pixels; cela dépend des caractéristiques du lien de communication entre les processeurs et la mémoire d'image. Une des premières réalisations de ce type d'algorithme a été effectuée sur la machine LINKS par Nishimura et al [20]. En France, des travaux effectués par le CCETT ont abouti à la réalisation de la machine CRISTAL. Cette machine multi-processeurs comporte 10 processeurs NS32032 disposant chacun de 512K de mémoire ainsi que d'un opérateur flottant. Sa topologie consiste en plusieurs grappes de processeurs connectées sur un bus commun. Un superviseur alloue à chaque processeur les pixels à calculer. Les performances de cette machine permettent d'obtenir des images d'une centaine de primitives en quelques dizaines de minutes et l'accélération reste linéaire en fonction du nombre de processeurs. Nous avons mis en œuvre ce type d'algorithme sur un hypercube iPSC/2. La figure 3 donne l'accélération en fonction du nombre de processeurs. Avec 32 processeurs, le temps de calcul reste élevé (1h30mn pour une image possédant 1801 objets à titre d'exemple).

2.4 Partition pour la subdivision

Cleary et al. [4] ont proposé une méthode qui consiste à découper de façon régulière le volume initial de la scène en plusieurs sous-volumes affectés chacun à un processeur. Les rayons issus de l'observateur sont envoyés aux processeurs possédant les informations né-

cessaires à leur traitement. Au cours du calcul, les rayons sont échangés entre processeurs quand ceux-ci quittent un sous-volume. Le processeur qui traite un pixel doit également recueillir les fractions d'intensités de ce pixel, calculées éventuellement par d'autres processeurs. Ceci est réalisé à l'aide de messages spécifiques qui peuvent transiter par un certain nombre de processeurs avant d'arriver au processeur chargé du pixel.

Le découpage régulier proposé par Cleary ne permet pas à chaque processeur d'avoir la même charge de travail. Celle-ci dépend à la fois du nombre de rayons traités par un processeur ainsi que du nombre d'objets qu'il possède et de leurs attributs géométriques et photométriques. Il n'existe pas de formule analytique donnant la complexité de traitement d'une .région de l'espace en fonction de ces paramètres pour toutes les images modélisables. Il est donc difficile d'associer à chaque processeur une région 3D, de telle façon que l'ensemble des processeurs soient utilisés au mieux.

Les travaux de Dippé et al. [12] et Nemoto et al. [19] tentent de résoudre ce problème en utilisant des techniques déjà employées dans le domaine du parallélisme.

Les deux solutions proposées ont pour principe de modifier en cours de calcul, et aussi souvent que nécessaire, la taille des régions associées aux processeurs afin de répartir dynamiquement la charge. Chaque processeur, en fonction de sa charge de travail et de celle de ses voisins dont il a connaissance par échange de messages, peut redistribuer une partie de sa charge. Cette redistribution se traduit par une modification de la taille des régions associées aux processeurs. Les objets contenus dans les régions redistribuées sont alors transférés aux processeurs correspondants. Les deux solutions proposées diffèrent par la forme des régions associées aux processeurs.

Si les solutions proposées sont efficaces pour obtenir une charge de travail équilibrée pour chaque processeur, elles n'ont pas été mises en œuvre, à cause de leur complexité, il est alors difficile de juger leur incidence sur l'efficacité globale de l'algorithme.

2.5 conclusion

Si la duplication des données dans chaque processeur permet d'obtenir une efficacité maximum sur une architecture parallèle, elle exige une taille mémoire importante dans chacun des processeurs. Cette efficacité obtenue est fictive, puisqu'elle est relative à un "mauvais" algorithme séquentiel. Dans un contexte de production audio-visuelle, les attributs photométriques, telles que les textures, nécessitent plusieurs dizaines de méga-octets. La duplication des données ne peut donc être envisagée dans ce cas, et seul le partitionnement des données permettrait la prise en compte de scènes complexes et réalistes. Par ailleurs de nouveaux modèles photométriques (lancer de rayon distribué [3, 9, 8, 7], radiosité [6]) augmentent les temps de synthèse et renforcent la nécessité de traitement parallèles. Nous avons donc choisi d'utiliser les techniques efficaces de subdivision spatiale et de répartir les données sur l'ensemble des processeurs afin de pouvoir traiter des scènes complexes. Dans le paragraphe suivant, nous décrivons des solutions adaptées au problème du partage de charge, de la terminaison et des interblocages. Les résultats d'une expérimentation sur un hypercube iPSC/2 sont également donnés.

3 Lancer de rayon sur un hypercube iPSC/2

3.1 L'hypercube iPSC/2

La machine mise à notre disposition est un hypercube iPSC/2 commercialisé par la société Intel résultant des recherches effectuées au Caltech [2]. Elle correspond à la deuxième génération d'hypercubes et diffère du premier modèle par l'adjonction d'un co-processeur de communication dans chaque nœud de l'hypercube permettant des routages plus efficaces.

L'iPSC/2 se présente sous la forme de plusieurs unités, chaque unité regroupant 32 cartes processeurs. La configuration disponible à l'IRISA comporte 64 processeurs. Le processeur numéro 0 est connecté à un processeur frontal (SRM) et sert d'intermédiaire pour les échanges entre les autres nœuds et le processeur frontal. L'ordinateur frontal est chargé de la compilation des programmes et de leur chargement dans les nœuds de l'hypercube. Le SRM est un PC/AT386 fonctionnant sous UNIX. Il permet à plusieurs utilisateurs de travailler simultanément sur la machine.

L'architecture d'un nœud est composée d'un processeur Intel 80386 à 16 MHz permettant de traiter les instructions à la vitesse de 4 MIPS. Un coprocesseur 80387 est utilisé pour les opérations flottantes et offre une performance de 300 KFLOPS. La mémoire disponible sur notre configuration est de 4 méga-octets par nœud pouvant aller jusqu'à 16 Mo. Les communications inter-processeurs sont gérés par un co-processeur spécialisé (DCM) assurant les fonctions de transfert et de routage des messages.

Un système d'exploitation de multiprogrammation réside sur chaque nœud. Le chargement des processus dans les nœuds peut être être fait globalement ou nœud par nœud, ce qui permet d'avoir des processus différents dans chacun des nœuds. Le système d'exploitation est accessible par l'intermédiaire d'une bibliothèque permettant, en particulier, le transfert de messages en mode synchrone ou asynchrone. Les langages de base disponibles sont C et FORTRAN. Chaque langage possède sa bibliothèque en fonction de ses particularités.

3.2 Partage équilibré des calculs

Concevoir un algorithme de lancer de rayon utilisant une répartition statique des données nécessite de répartir uniformément la charge de travail entre les processeurs. Cette charge dépend de la taille des sous-volumes associés au processeur. Comment déterminer a priori cette taille pour qu'au cours de l'exécution du programme de lancer de rayon, chaque processeur ait une charge de travail à peu près égale ? Nous tentons de répondre à cette question en utilisant la propriété de *cohérence des rayons.*

Cette propriété a fait l'objet d'études ayant pour but de minimiser les temps de calcul [16, 17, 22, 24]. La cohérence est le fait que deux rayons passant par des pixels voisins ont une grande probabilité d'intersecter les mêmes objets. Comme le montre la figure 4, cette propriété se retrouve également dans l'évaluation des rayons lumineux, transmis et réfléchis.

On peut donc penser que le temps de traitement d'un pixel est un bon estimateur du temps de traitement des pixels voisins. Nous utilisons cette hypothèse afin de déterminer un partitionnement correct du volume de la scène. La méthode de partitionnement que nous proposons consiste en trois phases distinctes :

1. découpage du volume de la scène en cellules à l'aide d'une grille 2D;

[2] California Institute of Technology

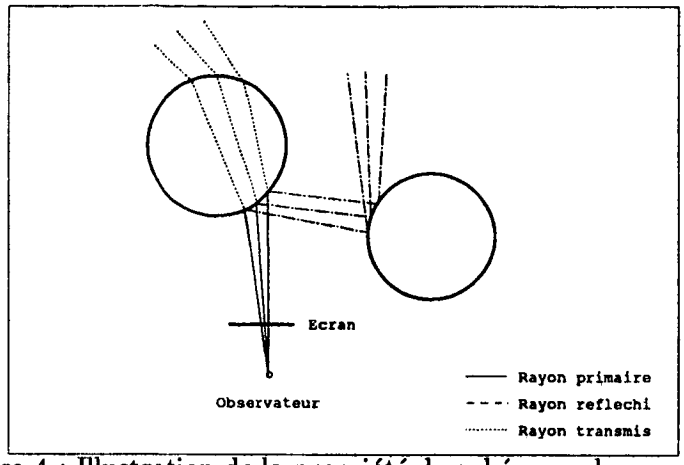

Figure 4 : Illustration de la propriété de cohérence des rayons

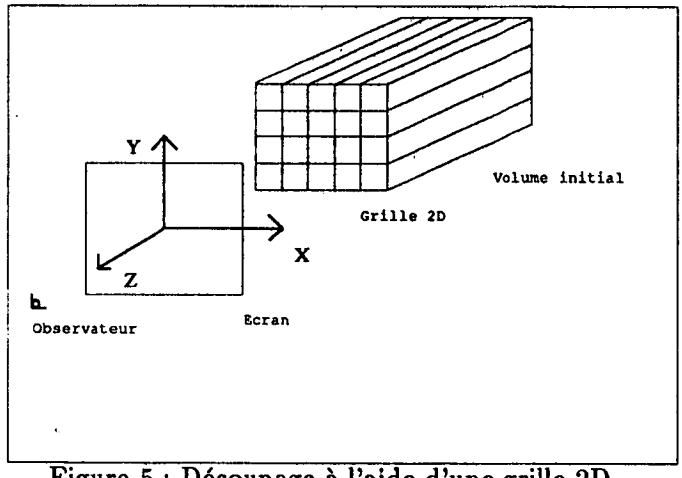

Figure 5 : Découpage à l'aide d'une grille 2D

2. sous-échantillonnage de l'image;

3. regroupement des cellules pour former des régions de complexité égale.

A l'issue de ces trois phases, l'espace de la scène est découpé en autant de régions que de processeurs disponibles dans la machine. Nous décrivons en détail ces trois phases dans les paragraphes suivants.

3.2.1 Découpage en cellules

Dans un premier temps le volume initial de la scène est découpé à l'aide d'une grille 2D suivant des plans perpendiculaires à celui de l'écran comme le montre la figure 5. Ce découpage crée un ensemble de sous-volumes identiques.

Chaque sous-volume ainsi créé est à nouveau découpé selon l'axe O_z afin d'obtenir un ensemble de cellules. Ceci permet de réduire le nombre d'objets contenus dans une cellule et augmente, par conséquent, l'efficacité de l'algorithme de lancer de rayon.

Il existe plusieurs manières de découper chaque sous-volume : soit de façon régulière (on obtient alors une grille 3D), soit en utilisant les volumes englobants des objets afin d'optimiser la gestion des espaces vides. Pour des raisons d'efficacité, nous avons opté pour la deuxième solution.

3.2.2 Sous-échantillonnage

La phase de découpage étant terminée, l'image est sous-échantillonnée, c'est à dire que l'on choisit de calculer un sous-ensemble de pixels parmi ceux constituant l'image. Ces pixels peuvent être obtenus en prenant un maillage plus ou moins fin de l'image : il suffit de prendre un pixel tous les K colonnes et tous les K lignes.

L'algorithme doit donc générer, pour chaque pixel de l'échantillon, un rayon issu de l'observateur. Pour chaque sous-volume on détermine le temps de traitement de tous les rayons entrant dans ce sous-volume. Chaque sous-volume est repéré par un couple (i, j). Le temps de traitement crédité par un rayon à un sous-volume (i, j) est la durée qui s'écoule entre l'instant où le rayon pénètre dans ce sous-volume et l'instant où il le quitte. A la fin de la phase de sous-échantillonnage, l'algorithme permet d'évaluer le temps de traitement total pour chaque sous-volume.

3.2.3 Regroupement en macro-région

L'information calculée à la phase 2 est utilisée dans la dernière phase afin de regrouper les sous-volumes en régions dont les temps de traitement (somme de tous les temps de traitement des sous-volumes les constituant) sont à peu près égaux. Le nombre de régions à créer est égal au nombre de processeurs disponibles dans la machine. La forme des régions, que nous appellons désormais *macro-régions*, est parallélépipèdique, ce qui permet de simplifier la détermination des macro-régions voisines. La technique de découpage que nous avons choisie est fortement liée au nombre et à l'organisation des 2^N processeurs d'un hypercube où N est la dimension de l'hypercube. Le regroupement en macro-régions est réalisé grâce à un découpage récursif d'un plan. Ce découpage est connu sous le nom de BSP (*Binary Space Partitionning*) [13]. Le plan que nous avons choisi est celui de la face avant du volume initial de la scène qui est parallèle au plan contenant l'écran.

Au premier niveau de récursivité (voir figure 6), l'algorithme découpe ce plan en deux parties, chacune de ces parties est à nouveau découpée en deux autres parties, et ainsi de suite. Au N^{ieme} appel, l'algorithme a créé 2^N parties du plan original. En utilisant les coordonnées sur l'axe z du volume initial, chacune de ces parties est étendue suivant cet axe pour former un volume parallélépipèdique. Pour effectuer ce découpage, nous utilisons successivement un segment horizontal puis vertical. Ce segment est choisi de telle façon que chaque plan créé contient la face avant de cellules dont la somme de tous les temps de traitement est à peu près égale pour un même niveau de récursivité. Cette disposition assure finalement que les temps de traitement associés aux macro-régions créées par cet algorithme sont quasiment identiques.

Les macro-régions ainsi créées n'ayant pas des tailles identiques, une macro-région peut être alors connexe à un nombre variable d'autres régions. La connexité des régions peut se modéliser par un graphe (appelé *graphe de connexité*) qui représente également les communications inter-processeurs possibles lors de l'exécution de l'algorithme de lancer de rayon. Ce graphe peut être utilisé pour associer les macro-régions au processeurs afin de minimiser le routage des communications.

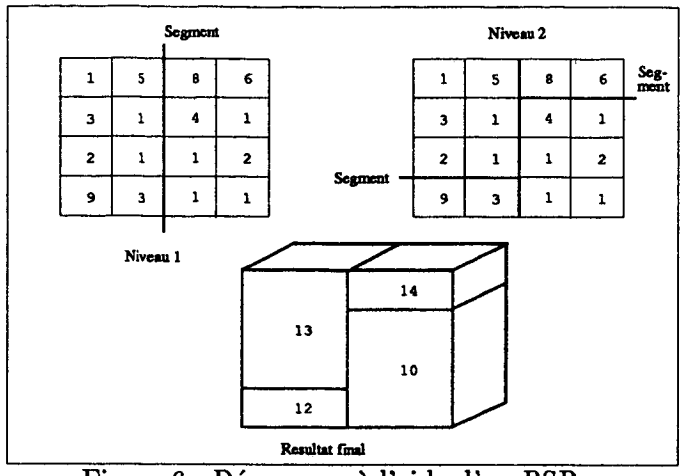

Figure 6 : Découpage à l'aide d'un BSP

3.2.4 Résultats

La figure 7 présente la corrélation entre deux échantillonnages réalisés sur une image de 128 × 128 pixels. Le premier consiste à choisir tous les pixels de l'image tandis que le deuxième est obtenu en ne prenant qu'un pixel tous les 8 lignes et 8 colonnes. L'axe des abscisses correspond au numéro d'un sous-volume, l'axe des coordonnéees donne un temps normalisé permettant la comparaison. Ces résultats montrent la forte corrélation qui existe entre les deux courbes. Les tests que nous avons effectués pour de nombreuses images, montrent qu'en général, un sous-échantillonnage représentant 1.5 % de l'image est suffisant pour réaliser une charge de calcul équilibrée pour l'ensemble des processeurs.

La figure 8 donne les temps de traitement associés à chaque macro-région. Le temps de traitement moyen d'une macro-région correspond à 70% du temps total de traitement de l'image. Ces résultats pourraient être améliorés si l'on disposait d'une horloge ayant une meilleure résolution pour effectuer le calcul du sous-échantillonnage. En effet, les calculs sont obtenus sur le frontal de l'iPSC dont l'horloge, permettant de mesurer le temps CPU, n'a qu'une résolution de 10 milli-secondes. Or un grand nombre de rayons ont un temps de traitement inférieur à cette résolution.

Ces résultats prouvent qu'un partitionnement a priori des données est possible grâce à la propriété de cohérence des rayons et est une bonne alternative aux solutions proposées par Dippé [11] et Nemoto [19]. Nous présentons dans le paragraphe suivant un algorithme parallèle de lancer de rayon utilisant cette technique.

3.3 Un aperçu du fonctionnement de l'algorithme

Le fonctionnement de l'algorithme sur l'hypercube peut se résumer en trois étapes qui s'ajoutent au prétraitement décrit précédemment.

1. Le frontal transmet à chaque processeur une macro-région ainsi que le sous-arbre CSG issu de la restriction de l'arbre initial. Il transmet également les informations de connexité permettant au processeur de déterminer quel processeur possède la macro-région dans la direction d'un rayon lorsque ce dernier quitte la macro-région associée au processeur.

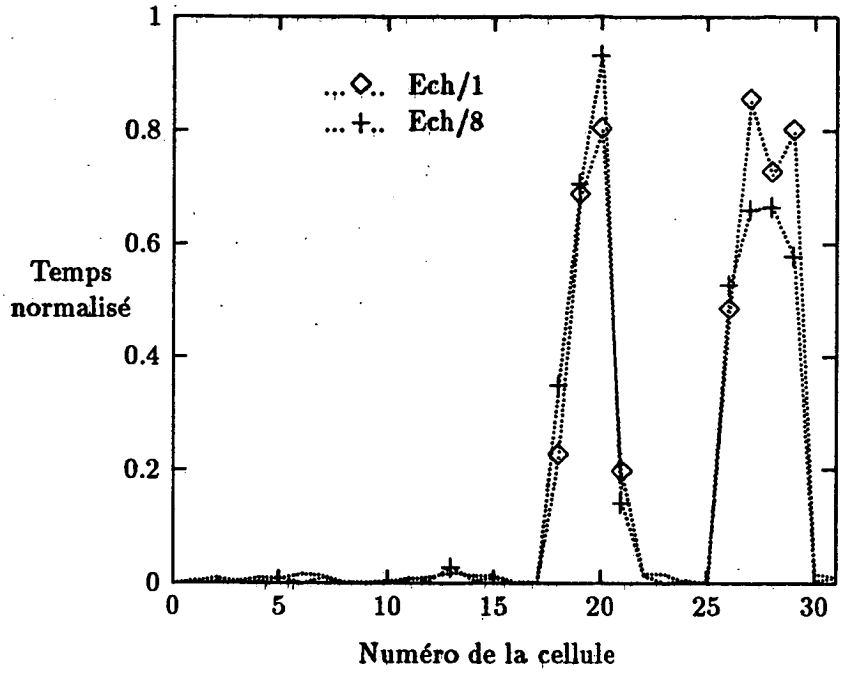

Figure 7 : Corrélation entre les sous-échantillonnages

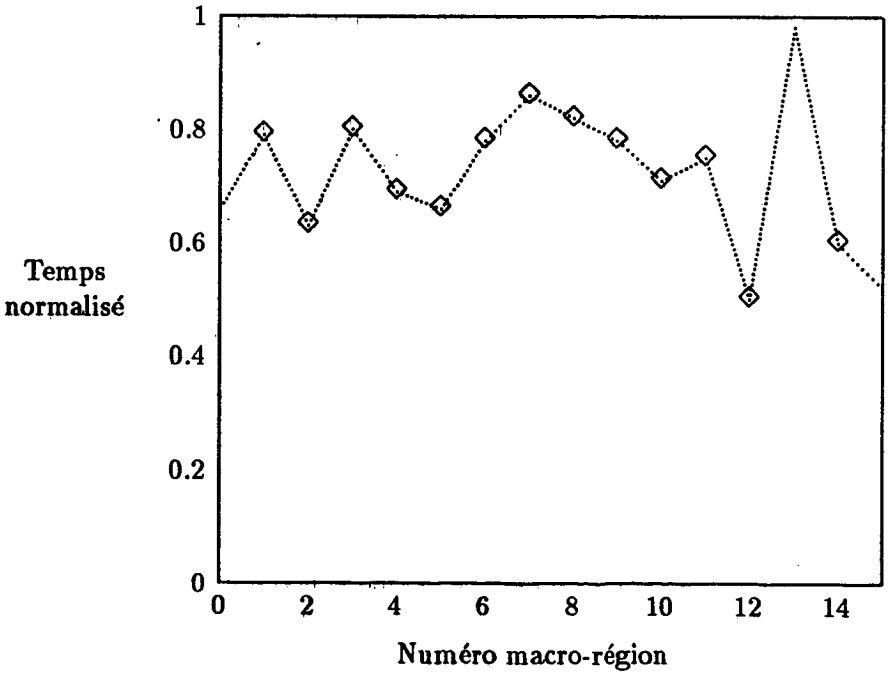

Figure 8 : Temps de traitement des macro-régions

2. Chaque processeur effectue une subdivision spatiale de la macro-région qu'il a reçue. A la fin de cette étape, un message est envoyé au processeur frontal pour lui indiquer que l'étape de subdivision est terminée. Quand le processeur frontal a reçu ce message de tous les processeurs, il autorise ceux-ci à débuter la phase de synthèse.

3. Cette dernière étape, ou tâche de synthèse, consiste à calculer les pixels de l'image. Chaque processeur s'occupe d'un sous-ensemble de pixels. Ce sous-ensemble dépend à la fois de la position et de la taille de la macro-région associée au processeur par rapport à l'écran. Il peut donc être vide, dans ce cas le processeur a pour rôle de ne consommer que les rayons provenant des autres processeurs.

Le fonctionnement de la tâche de synthèse peut être résumé par l'algorithme suivant (pseudo-C) :

```
main()
{
   /* Traitement des rayons primaires */

   while (generation_rayon_primaire(r)) {

      evaluation_rayon(r);

      /* Traitement des rayons provenant  */
      /* des autres processeurs           */

      while(rayon_a_lire()) {
         lire_rayon(r);
         calcul_region_entree(r);
         evaluation_rayon(r);
      }
   }

   /* Il n'y a plus de rayon primaire */

   /* L'algorithme a-t-il termine ? */

   while(image_non_terminee()) {

      /* Traitement des rayons provenant  */
      /* des autres processeurs           */

     while (rayon_a_lire()) {
        lire_rayon(r);
        calcul_region_entree(r);
        evaluation_rayon(r);
     }
   }
}
```

generation_rayon_primaire(r) rend vrai si il existe encore un rayon primaire à générer.

evaluation_rayon(r) évalue un rayon en fonction de son type. Des rayons secondaires et lumineux peuvent être émis.

rayon_a_lire() rend vrai si un rayon a été reçu.

lire_rayon(r) lit le rayon provenant d'un autre processeur.

calcul_region_entree(r) détermine la région d'entrée d'un rayon provenant d'un autre processeur.

image_non_terminee() met en œuvre l'algorithme de détection de la terminaison.

3.3.1 Détection de la terminaison

La fonction *image_non_terminee()*, qui indique si un processeur a terminé, met en œuvre un algorithme distribué. En effet, chaque processeur ne peut déterminer seul s'il a terminé son travail car il peut recevoir à tout moment un rayon à calculer envoyé par un processeur voisin. Pour détecter la terminaison de l'algorithme, nous utilisons l'algorithme du jeton proposé par Dijkstra [10]. Pour ce faire, un anneau virtuel est créé reliant l'ensemble des processeurs de la machine et sur lequel circule un jeton ayant une couleur noire ou blanche. A l'initialisation, le processeur 0 possède le jeton de couleur blanche. Un processeur P recevant le jeton se comporte selon les règles suivantes:

1. Si le jeton est noir, le processeur P le transmet au processeur suivant dans l'anneau.
2. Si le jeton est blanc, il est transmis blanc si le processeur P est inactif et n'a pas envoyé de message à un processeur le précédant dans l'anneau depuis le dernier passage du jeton. Sinon P change en noir la couleur du jeton avant de le transmettre au processeur suivant.

Si après un tour de l'anneau, le jeton est resté blanc, la terminaison de l'algorithme est alors effective. Sinon, le processeur 0 émet un nouveau jeton de couleur blanche. Les propriétés de l'algorithme de synthèse nous permettent d'effectuer une mise en œuvre efficace de cet algorithme de détection de la terminaison. Dans le processus de synthèse, nous pouvons dégager, en effet, deux étapes : la première consiste à générer les rayons primaires tout en consommant les rayons provenant des autres processeurs, la deuxième étape débute quand il n'y a plus de rayons primaires à calculer. A ce stade, le processeur a potentiellement terminé son travail et il ne lui reste qu'à consommer les rayons provenant des autres processeurs. C'est dans cette étape que le jeton va circuler de processeur en processeur sur l'anneau virtuel. Chaque processeur ne fait circuler le jeton que s'il est lui même parvenu à cette étape. Le nombre de tours effectués par le jeton pour détecter la terminaison effective de l'algorithme est ainsi réduit. Les premiers résultats ont montré qu'une vingtaine de tours est nécessaire pour détecter la terminaison de l'algorithme.

Une difficulté vient du fait que l'algorithme de Dijkstra fait l'hypothèse de l'instantanéité des transferts de messages. Une mise en œuvre directe sur une architecture de type hypercube telle que l'iPSC/2 peut donc entraîner une fausse détection de la terminaison car cette propriété n'est pas respectée. L'exemple suivant montre le cas où l'algorithme de Dijkstra conclut à la terminaison alors que des messages sont encore en transit.

Un processeur P_i transmet un message *msg* à P_{i+k} qui est un successeur de P_i dans l'anneau. Puis il envoie le jeton dont la couleur est blanche. Si P_{i+k} reçoit le jeton avant le message, la terminaison de l'algorithme pourra alors être détectée alors que l'algorithme n'est pas terminé puisqu'il reste encore un message à recevoir. Nous avons cependant

Distance	% temps
1	0 %
2	4 %
3	7 %
4	12 %
5	16 %

Figure 9 : Surcoût lors d'un transfert entre processeurs

implanté directement l'algorithme en nous basant sur les vitesses de transferts des messages en fonction de la distance parcourue. Pour des messages de taille identique, le tableau 9 donne le pourcentage de temps supplémentaire pour envoyer un message d'un processeur à un autre suivant la distance qui les séparent (distance calculée en fonction du numéro de processeur dans le code de GRAY).

La topologie en anneau, nécessaire nécessaire à la mise en œuvre de l'algorithme de terminaison, s'implante directement sur la topologie hypercube, c'est à dire que la distance entre deux processeurs voisins dans l'anneau est 1. Dans le cas de l'exemple précédent, le jeton ne pourra donc pas être reçu par le processeur P_{i+k} avant la réception du message *msg*. En effet, si le temps nécessaire à l'échange d'un message par deux processeurs adjacents est t, celui mis par deux processeurs non directement connectés pour échanger un message est au maximum égal à $t+0.16t$ pour un hypercube de dimension 5. Le processeur $P_{i+k}, (k \geq 2)$ ne peut pas recevoir le jeton avant le message *msg* car il ne recevra le jeton qu'au bout d'un temps minimum de $2t$.

Si nous prenons une taille de message contenant un jeton égale à la taille d'un message contenant un rayon, nous pouvons appliquer directement l'algorithme de Dijkstra.

3.3.2 Les interblocages potentiels

La mise en œuvre de la partie communication de notre algorithme utilise des primitives de communication non bloquantes. Les processeurs ne peuvent donc se bloquer en attente de réception de messages qui ne sont pas produits. Les seules situations d'interblocage proviennent de la saturation de la file d'attente associée à chaque processeur où sont enregistrés à la fois les messages destinés à être émis et les message reçus par le processeur. Lorsqu'un processeur P_i a saturé sa file d'attente, il ne peut recevoir et traiter des rayons provenant des autres processeurs. Si P_j veut émettre des rayons vers P_i, la file d'attente de P_j deviendra également saturée et ceci provoquera un interblocage. Ce problème a été résolu en utilisant le processeur frontal pour enregistrer provisoirement les rayons qui ne peuvent être stockés par les processeurs réellement destinataires, ceux-ci ayant leurs files d'attente saturées. Lorsque ces processeurs n'ont plus de rayons primaires à produire, le frontal leur transmet les rayons qui leur étaient destinés. En ajustant la taille des files d'attente dans chaque processeur, nos expériences ont montré que le frontal recevait peu de rayons par rapport au nombre total de rayons générés par l'ensemble des processeurs. La capacité du frontal peut donc être estimée suffisante pour remplir dans tous les cas sa fonction de stockage intermédiaire.

3.4 Résultats

Nos expériences sur l'hypercube ont consisté à tester deux algorithmes. Le premier duplique une hiérarchie de volumes englobants tandis que le deuxième utilise la méthode

Image	Résolution	Sources Lumineuses	Profondeur	Primitives
rings1	256 x 256	3	2	30 cylindres, 1 parallélépipède, 30 sphères
rings2	256 x 256	3	2	150 cylindres, 1 parallélépipède, 150 sphères
rings3	256 x 256	3	2	420 cylindres, 1 parallélépipède, 420 sphères
rings4	256 x 256	3	2	900 cylindres, 1 parallélépipède, 900 sphères

Figure 10 : Caractéristiques des images

décrite dans les paragraphes précédents. Nous les avons exécutés pour des images composées de 61 à 1801 primitives volumiques incluant des sphères, cylindres et parallélépipèdes. Le tableau 10 décrit les caractéristiques de ces images.

Dans un premier temps nous avons comparé le comportement de ces deux algorithmes parallèles lorsque le nombre d'objets augmente. Comme le montre la figure 11, les résultats sont tout à fait similaires à ceux obtenus lors d'une exécution séquentielle (figure 1). Le deuxième algorithme conserve un temps constant pour des images contenant 61 à 1801 primitives.

La figure 12 et 13 montre les facteurs d'accélération obtenus sur les quatre images tests pour un nombre de processeurs croissant. Les accélérations obtenues ne sont pas linéraires en fonction du nombre de processeurs. Des expériences sur 64 processeurs mettent en évidence, dans le cas de l'algorithme de partitionnement, qu'il est de moins en moins efficace dès que le nombre de processeur est supérieur à la trentaine. Elle ne peut donc être envisagée sur des architectures massivement parallèles possédant plusieurs centaines de processeurs.

La cause essentielle de cette inefficacité est l'accroissement du volume de calcul quand le nombre de macro-régions augmente. Il existe deux causes à cet accroissement de calcul :

1. Quand un rayon est reçu par un processeur, celui-ci doit déterminer la cellule d'entrée avant de continuer le traitement du rayon. Ce calcul n'existe que parce qu'il y a découpage en macro-régions. Donc plus il y a de macro-régions, plus l'algorithme parallèle est pénalisé.

2. Quand le nombre de processeurs s'accroit, la probabilité qu'un objet appartienne à la fois à deux macro-régions (donc traité par deux processeurs) entraîne un accroissement du nombre d'intersections entre rayons et primitives volumiques. C'est un défaut majeur des techniques de subdivision spatiale. Une solution a été décrite par Arnaldi [1] mais dans un contexte d'une exécution sur une machine séquentielle disposant d'une mémoire commune. Sa mise en œuvre sur une machine à mémoire distribuée ne permet pas d'obtenir la même efficacité.

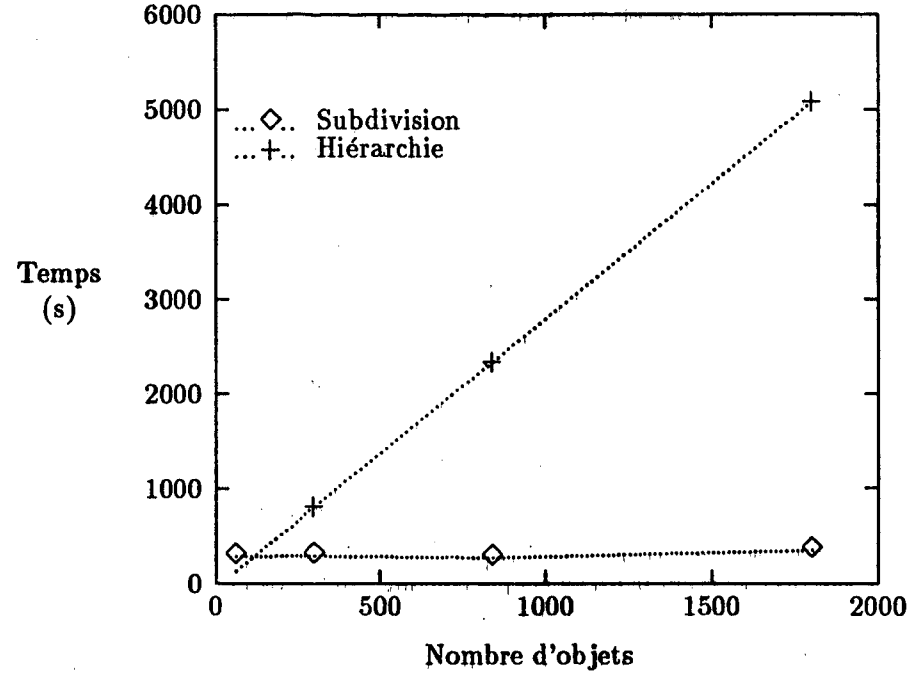

Figure 11 : Comparaison subdivision-hiérarchie (32 processeurs)

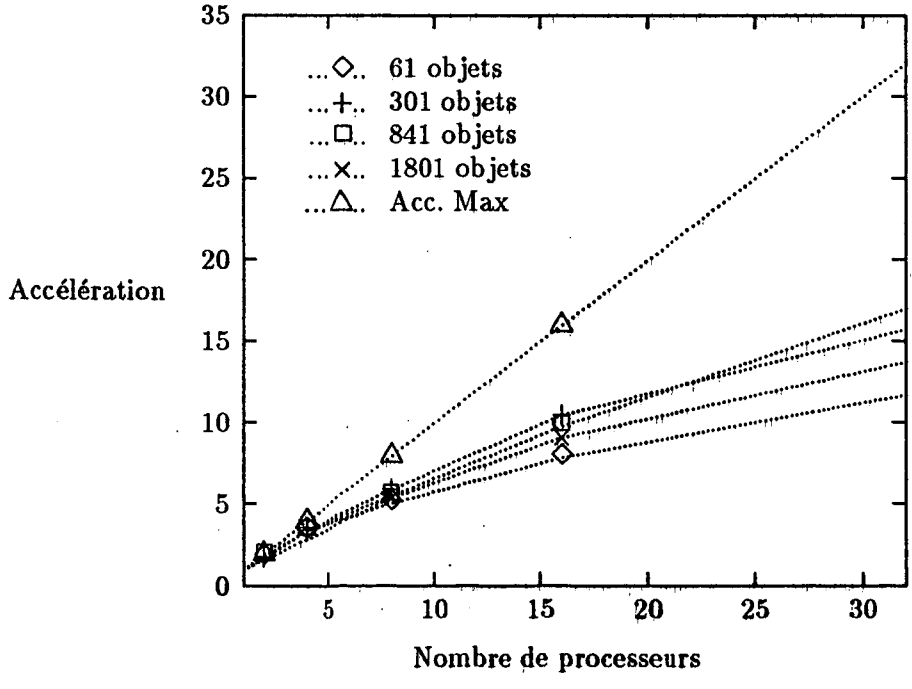

Figure 12 : Accélération de l'algorithme de lancer de rayon avec partition des données

Processeurs	Temps (s)	Accélération	Efficacité
1	4578	1.00	1.00
2	2392	1.91	0.96
4	1347	3.39	0.85
8	827	5.54	0.69
16	468	9.78	0.61
32	269	17.01	0.53

Figure 13 : Temps d'exécution

4 Conclusion

Un algorithme parallèle de synthèse d'images par lancer de rayon a été présenté. Un des problèmes majeurs rencontré lors de sa mise en œuvre a été de déterminer correctement la taille de chaque sous-volume afin d'obtenir une distribution équitable des calculs sur l'ensemble des processeurs. Les techniques de distribution dynamique des calculs ne peuvent être employées sur ce type d'algorithme si l'on veut rester dans un contexte de production audio-visuelle car elles nécessitent le transfert de données de taille importante entre processeurs. Nous avons donc proposé une méthode permettant de calculer statiquement la taille de ces sous-volumes par un sous-échantillonnage de l'image. Par ailleurs, la mise en œuvre de l'algorithme de synthèse nous a permis d'expérimenter des algorithmes distribués proposés pour résoudre des problèmes classiques telle que la détection de la terminaison. Grâce à la connaissance précise de l'architecture support et de l'application, nous avons pu trouver des techniques de réalisations efficaces.

Bibliographie

[1] B. Arnaldi, T. Priol, and K. Bouatouch. A new space subdivision method for ray tracing csg modelled scenes. *The Visual Computer*, 3(2):98–108, August 1987.

[2] C. Bouville, R. Brusq, J.L. Dubois, and I. Marchal. Synthèse d'images par lancer de rayons: algorithmes et architecture. In *Premier Colloque Image*, pages 683–696, May 1984.

[3] C. Bouville, J.L. Dubois, I. Marchal, and M.L. Viaud. Monte-carlo integration applied to an illumination model. In *EUROGRAPHICS'88 Conference Proceeding*, pages 483–498, September 1988.

[4] J.G Cleary, B. Wyvill, G.M. Birtwistle, and R. Vatti. *Multiprocessor Ray Tracing.* Research Report 83/128/17, University of Calgary, October 1983.

[5] J.G Cleary, B.M Wyvill, G.M. Birtwistle, and R. Vatti. Multiprocessor ray tracing. *Computer Graphics Forum*, 5(1):3–12, March 1986.

[6] M. Cohen and D. Greenberg. The hemi-cube, a radiosity solution for complex environments. *ACM Computer graphics*, 19(3), 1985.

[7] R.L Cook. Practical aspects of distributed ray tracing. SIGGRAPH'86 Developments in Ray Tracing seminar notes.

[8] R.L. Cook. Stochastic sampling in computer graphics. *ACM Transaction on Graphics*, 5(1):51–72, January 1986.

[9] R.L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *SIGGRAPH'84*, pages 137–145, ACM, July 1984.

[10] E.W Dijkstra, W.H.J Feijen, and A.J.M Van Gasteren. Derivation of a termination detection algorithm for distributed computation. *Inf. Proc. Letters*, 16:217–219, June 1983.

[11] M. Dippe and J. Swensen. An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. *Computer Graphics*, 18(3):149–158, July 1984.

[12] M.A.Z. Dippé and E.H Wold. Antialiasing through stochastic sampling. *ACM Computer Graphics*, 19(3), 1985.

[13] H. Fuchs. On visible surface generation by a priori tree structure. In *SIGGRAPH'80 Conference Proceeding*, pages 149–158, July 1980.

[14] A. Fujimoto, T. Tanaka, and K. Iawata. Arts : accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, April 1986.

[15] A. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[16] P.S Heckbert. Beam tracing polygonal objects. In *SIGGRAPH'84*, pages 119–127, ACM, July 1984.

[17] J.T. Kajiya. Raytracing parametric patches. *IEEE Computer Graphics and Applications*, 16(3):245–254, March 1982.

[18] M. R. Kaplan. Space-tracing, a constant time ray tracer. In *SIGGRAPH'85 tutorial on the uses of spatial coherence in ray tracing*, 1985.

[19] K. Nemoto and T. Omachi. An adaptative subdivision by sliding boundary surfaces for fast ray tracing. *Graphics Interface*, 43–48, 1986.

[20] H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa, and K. Omuira. Links-1: a parallel pipelined multimicrocomputer system for image creation. In *Proc. of the 10th Symp. on Computer Architecture*, pages 387–394, 1983.

[21] S.D Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18(2):109–144, February 1982.

[22] L.R. Speer, T.D. DeRose, and B.A. Barsky. A theoretical and empirical analysis of coherent ray-tracing. In *Graphics Interface'85*, pages 11–25, May 1985.

[23] M. Tamminen, O. Koronen, and M. Mantyla. Ray-casting and block model conversion using a spatial index. *CAD*, 16(4):203–208, July 1984.

[24] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23:343–349, June 1980.

LISTE DES DERNIERES PUBLICATIONS INTERNES

I 447 **VISION 3D : UNE NOUVELLE METHODE DE LA TRIANGULATION POUR LA VISION MONOCULAIRE OU POLYNOCULAIRE DANS UNE SEQUENCE D'IMAGES**
Ming XIE, Patrick RIVES
48 Pages, Janvier 1989.

PI 448 **DYNAMIC VISION : DOES 3D SCENE PERCEPTION NECESSARILY NEED TWO CAMERAS OR JUST ONE ?**
Ming XIE
38 Pages, Janvier 1989.

PI 449 **CONVOLUTION SYSTOLIQUE DE FONCTIONS ARITHMETIQUES**
Patrice QUINTON, Yves ROBERT
30 Pages, Janvier 1989.

PI 450 **MISE EN OEUVRE D'ALGORITHMES NUMERIQUES SUR UN HYPERCUBE**
Brigitte VITAL
28 Pages, Janvier 1989.

PI 451 **LANCER DE RAYON SUR DES ARCHITECTURES PARALLELES : UNE ETUDE DE PERFORMANCE**
Thierry PRIOL, Kadi BOUATOUCH
20 Pages, Janvier 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique